

JRun Tag Library Reference

Copyright Notice

© 2000 Allaire Corporation. All rights reserved.

This manual, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. The content of this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Allaire Corporation. Allaire Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Allaire Corporation.

ColdFusion and HomeSite are federally registered trademark of Allaire Corporation. Allaire, Allaire Spectra, JRun, <CF_Anywhere>, the ColdFusion logo, the JRun logo, and the Allaire logo are trademarks of Allaire Corporation in the USA and other countries. Microsoft, Windows, Windows NT, Windows 95, Microsoft Access, and FoxPro are registered trademarks of Microsoft Corporation. Java, JavaBeans, JavaServer, JavaServer Pages, JavaScript, JDK, and Solaris are trademarks of Sun Microsystems Inc. UNIX is a trademark of The Open Group. PostScript is a trademark of Adobe Systems Inc. All other products or name brands are the trademarks of their respective holders.

Contents

Chapter 1: Introduction to the JRun Tag Library.....	1
Scope.....	2
Design Goals Principles.....	2
Chapter 2: Custom Actions	5
Conventions	6
Multiple object types	7
Tag summary.....	7
Sql.....	9
SqlParam.....	15
SendMsg.....	17
MsgParam	20
GetMsg	21
Transaction.....	24
Jndi	25
SendMail	29
MailParam	33
GetMail.....	35
Servlet.....	40
ServletParam	41
Query2Xml.....	42
Xslt	44
Form	46
Input.....	48
Select	51
Param	54
ForEach	56
If.....	59
Switch.....	60
Case	61

CHAPTER 1

Introduction to the JRun Tag Library

The JRun Tag Library is a collection of custom actions based on the JavaServer Pages 1.1 (JSP) tag extension API. These custom actions (also called custom tags or tags) allow JSP developers to develop Web applications that incorporate sophisticated client-side features and implement server-side Java 2 Enterprise Edition (J2EE) technologies.

For example, by using the JRun Tag Library, JSP developers can perform data queries, transactions, asynchronous message sending/receiving, e-mail processing, or XML data processing without knowing all the APIs involved. The JRun custom tags seamlessly integrate JSP with the associated leading-edge J2EE technologies to give developers the ability to complete tasks without writing a single line of Java code in their JSP pages. The JRun custom tags also provide sophisticated, yet simple and elegant mechanisms to generate presentation code for desktop browser software, narrow-band wireless devices, and PDAs.

Contents

• Scope	2
• Design Goals Principles	2

Scope

The JRun Tag Library can be categorized into 3 main parts:

- J2EE technologies
- Client-side form processing
- Flow control

For a list of JRun custom tags, refer to “Custom Actions” on page 5.

Design Goals Principles

The JRun Tag Library was developed with the following design goals.

- Provide tag abstractions to commonly-used Java APIs and server-side processing.

A common problem that many JSP developers encounter is writing too many lines of Java code in JSP pages, which makes the JSP pages hard to read and maintain. Often a JSP developer has to study and understand a variety of Java APIs in order to accomplish something that the standard JSP specification does not provide. The traditional JSP/JavaBean approach does not always work or does not look elegant when there is a need for inter-bean communications. The JSP/custom tag approach solves this fundamental problem by providing a tag-based scripting look-and-feel, inter-tag communication, and parent-child tag communication.

J2EE outlines the most commonly used Java APIs for Web application development. The JRun Tag Library allows a JSP developer to perform most tasks without writing any Java code.

One example of the JRun Tag Library abstracting complex processing is in the case of database queries. Without tag extensions, a JSP developer must write JDBC code in a Java scriptlet to perform the required database query and then display the result set with highly formatted presentation code. This results in lengthy and error-prone development cycles. This situation becomes worse if a JSP developer is required to write code to use different Java API implementations.

- Keep the tag syntax simple and easy to understand.

While a tag abstraction cannot always cover every aspect of a particular Java API, The JRun Tag Library strikes a balance between providing tags that are broad enough to cover the most important part of an API while still providing a simple and easy-to-understand tag syntax.

- Provide portability.

The JSP specification ensures that JavaServer Pages work exactly the same way within JSP containers from different vendors. When a standard tag library specification is finalized, JSP pages written using custom tags should also work the same way within JSP containers from different vendors. To maximize portability, the JRun Tag Library conforms strictly to all relevant J2EE requirements.

- Provide support for industry standards where possible.

In order to achieve maximum portability, the JRun Tag Library uses standard APIs whenever possible. A tag library should be designed and implemented so that it is highly compatible with standard Java APIs.

- Allow for vendor-independent hooks to improve performance.

Since custom tags are used in JSP (run as servlets on top of an application server infrastructure) it is very important to define and implement a tag library so that custom tags can take full advantage of an application server's abilities. For example, the JRun Tag Library enhances performance by supporting database connection pooling and result set caching, if such features are available in the application server.

CHAPTER 2

Custom Actions

This chapter provides detailed descriptions of the JSP custom actions in the JRun Tag Library.

Contents

• Conventions	6
• Multiple object types	7
• Tag summary	7
• Sql.....	9
• SqlParameter	15
• SendMsg.....	17
• MsgParam	20
• GetMsg.....	21
• Transaction	24
• Jndi	25
• SendMail	29
• MailParam.....	33
• GetMail.....	35
• Servlet.....	40
• ServletParam.....	41
• Query2Xml.....	42
• Xslt.....	44
• Form	46
• Input.....	48
• Select	51

• Param	54
• ForEach	56
• If	59
• Switch	60
• Case	61

Conventions

Each tag description includes the following sections, where applicable:

- Syntax
- Usage
- Attributes
- Tag Library Descriptor
- Scripting Variable
- Tag Interface (for parent-child tag communication)
- Example

For more information on tag libraries, refer to the *Developing Applications with JRun* manual.

Multiple object types

Where possible, tag attributes in the JRun Tag Library support multiple object types. For example, The foreach tag's group attribute supports `java.util.Enumeration` and `java.util.Iterator`, etc. In addition, some tag attributes support convenient notations:

- If a tag attribute requires an object type other than `java.lang.String`, the syntax will generally be in the form `attribute="<%< x %>"`, where `x` is an object or a method that returns an object. The above syntax is required for scripting variables generated in Java scriptlets. If this non-string object can be found in the `pageContext` object, the syntax can be simplified to `attribute="scope.name"`, as shown in the following example:

```
<foo id="x" scope="page" ... />
<foo id="y" scope="request" .../>
<% Object z = new Object(); %>
<bar attr1="page.x" attr2="request.y" attr3="<%<= z %>" .../>
```

- If a tag attribute requires a primitive scripting variable, the syntax will generally be in the form `attribute="<%<= x %>"`, where `x` is a primitive scripting variable or a method that returns a primitive scripting variable. When using the JRun Tag Library, the syntax can be simplified as shown in the following example:

```
// Instead of:
<foo attr="<%<= 123 %>" .../>
// You can use:
<foo attr="123" .../>
```

- If a tag attribute requires an object that is usually bound to a JNDI context, such an object can be retrieved automatically by specifying the short name of the full JNDI lookup name. For example, if there is a data source bound to the JRun JNDI context named "java:comp/env/jdbc/source1", the `sql` tag's `datasrc` attribute can take: `datasrc="source1"`. This technique also applies to other J2EE subcontexts: `java:comp/env/jdbc`, `java:comp/env/jms`, `java:comp/env/mail`, `java:comp/env/ejb` and `java:comp/env/url`.

Tag summary

The following is a summary of the tags described in this chapter:

JRun Tag Summary	
Tag name	Description
Sql	Performs database queries.
SqlParameter	Used with <code>sql</code> to dynamically construct SQL statements.
SendMsg	Sends asynchronous messages.

JRun Tag Summary (continued)

Tag name	Description
MsgParam	Used with <code>sendmsg</code> to set message properties.
GetMsg	Receives asynchronous messages.
Transaction	Performs distributed transactions.
Jndi	Looks up objects and directories (for example, EJB, LDAP, and CORBA).
SendMail	Sends e-mail messages.
MailParam	Used with <code>sendmail</code> to set additional mail headers and construct multipart e-mails (for example, e-mail attachments).
GetMail	Receives e-mail messages.
Servlet	Invokes java servlets.
ServletParam	Used with <code>servlet</code> to set servlet attributes.
Query2Xml	Converts database query results into XML documents.
Xslt	Performs XSLT transformations.
Form	Generates an HTML form with client-side JavaScript for form validation.
Input	Generates HTML input statements with client-side JavaScript for input field validation.
Select	Generates HTML select and option statements with client-side JavaScript for selection validation.
Param	Declares scripting variables.
ForEach	Loops over a collection of objects.
If	Performs conditional block execution
Switch	Performs conditional block execution similar to the switch-case construct in Java
Case	Used with <code>switch</code> to perform conditional block execution

Sq1

Syntax <sql ...>
 sql statement with optional <sqlparam .../>
 ...
</sql>

Usage Performs database operations by sending the enclosed SQL statement to the specified data source. It is possible to have flow-control tags within the `sql` tag so that the SQL statement can be dynamically constructed. The `sql` tag can also be used with `sqlparam` tags to construct parameterized SQL statement, for example, inserting binary objects into the SQL statement. If the `sql` tag is within a `transaction` tag, the `sql` tag will be a transaction participant and the success or failure of the `sql` operation will be based on all the operations within the same transaction. If all the operations (including `sql`, `sendmsg` and `getmsg`) within the `transaction` are successful, all the changes will be committed. Otherwise, an exception will be raised and changes will be rolled back. The `sql` tag uses the `allaire.taglib.Transaction` interface to detect if it is enclosed in a `transaction` tag.

The `sql` tag supports three types of syntax. Connection pooling is supported when a `DataSource` is provided. The J2EE server administrator should predefine data source objects with connection pooling support to maximize Web application performance. For details, refer to the J2EE specification and the following examples.

Syntax Types

Syntax Type 1: JDBC connection object (`java.sql.Connection`) is provided.

Syntax Type 2: J2EE data source (`javax.sql.DataSource`) is provided.

Syntax Type 3: JDBC driver class and url to the database are provided.

The following table shows which attributes are available using the three different syntax types:

Sql tag Syntax Types			
Attribute	Type 1	Type 2	Type 3
connection	x		
id	x	x	x
scope	x	x	x
datasrc		x	
username		x	x
password		x	x

Sql tag Syntax Types (continued)			
Attribute	Type 1	Type 2	Type 3
driver			x
url			x

Attributes **connection (type 1)**

Required. Takes either `java.sql.Connection` or `java.lang.String` as an attribute. If a string is specified, it is assumed that the `connection` object can be obtained by invoking `pageContext.getAttribute()`. Since the `sql` tag is provided with a JDBC connection, the tag will return the connection object without closing it.

id (types 1,2,3)

Optional. Takes `java.lang.String` as an attribute. The string name is used as the scripting variable name for the query result. This attribute can be omitted if the SQL statement doesn't return a result set.

scope (types 1,2,3)

Optional. Takes `java.lang.String` as an attribute. Valid string values are `page`, `request`, `session`, and `application`. The default is `page`.

datasrc (types 2)

Required. Takes `javax.sql.DataSource` or `java.lang.String` as an attribute. If a string is specified, it is assumed that the datasource object can be obtained by performing a JNDI lookup with "`java:comp/env/jdbc/[datasrc]`".

To use the `sql` tag in a distributed transaction, you must use this attribute (type 2 syntax) and it must be of type `java.lang.String`.

username (types 2,3)

Optional. Takes `java.lang.String` as an attribute. The string value is used as the username for the data source.

password (types 2,3)

Optional. Takes `java.lang.String` as an attribute. The string value is used as the password for the data source.

driver (types 3)

Required. Takes `java.lang.String` as an attribute. The string value is the JDBC driver class name.

url (types 3)

Required. Takes `java.lang.String` as an attribute. The string value is the JDBC URL to the database.

```
TagLib Descriptor <tag>
    <name>sql</name>
    <tagclass>allaire.taglib.SqlTag</tagclass>
    <teiclass>allaire.taglib.SqlTei</teiclass>
    <bodycontent>JSP</bodycontent>
    <attribute>
        <name>id</name>
        <required>false</required>
        <rtpvalue>false</rtpvalue>
    </attribute>
    <attribute>
        <name>scope</name>
        <required>false</required>
        <rtpvalue>false</rtpvalue>
    </attribute>
    <attribute>
        <name>driver</name>
        <required>false</required>
        <rtpvalue>true</rtpvalue>
    </attribute>
    <attribute>
        <name>url</name>
        <required>false</required>
        <rtpvalue>true</rtpvalue>
    </attribute>
    <attribute>
        <name>datasrc</name>
        <required>false</required>
        <rtpvalue>true</rtpvalue>
    </attribute>
    <attribute>
        <name>connection</name>
        <required>false</required>
        <rtpvalue>true</rtpvalue>
    </attribute>
    <attribute>
        <name>username</name>
        <required>false</required>
        <rtpvalue>true</rtpvalue>
    </attribute>
    <attribute>
        <name>password</name>
        <required>false</required>
        <rtpvalue>true</rtpvalue>
    </attribute>
```

```
</tag>
```

Scripting Variables The `sql` tag optionally stores database query result in the `pageContext` object. If `id` is specified, an instance of `allaire.taglib.QueryTable` will be stored. The scope of this object depends on the `scope` attribute, which is `PageContext.PAGE_SCOPE` by default. The following is the `allaire.taglib.QueryTable` API:

```
package allaire.taglib;

public class QueryTable extends Table {
    public QueryTable()
    public String[] Names;
    public Object[] Values;
    public void populate(ResultSet rs) throws SQLException;
    public boolean next();
}

public class Table {
    public Table()
    public boolean next();
    public void resetCursor();
    public String get(int index) throws Exception;
    public String get(String column) throws Exception;
    public Object getObject(int index) throws Exception;
    public Object getObject(String column) throws Exception;
    public String getColumnLabel(int index);
    public int getRowCount();
    public int getColumnCount();
    public String[] getColumnNames();
    public Table nextTable();
    public Table previousTable();
    public Table firstTable();
    public Table lastTable();
    public void setNextTable(Table t);
    public void setPreviousTable(Table t);
    public String toString();
}
```

The API supports database queries that return multiple result sets by chaining tables together. Also, for each `next` method call, the field values are available as an `Object` array instead of calling `getObject` for individual values.

Interface

```
package allaire.taglib;

public interface Sql {
    void setSqlParam(Object value);
    void setSqlParam(Object value, int sqltype);
    void setSqlParam(Object value, int sqltype, int scale);
}
```

Example This example shows three different uses of the `sql` tag.

```
<%@ page import="java.sql.* , javax.sql.* , allaire.taglib.*" %>
<%@ taglib uri="jruntags" prefix="jrun" %>

<%
Class.forName("....").newInstance();
Connection con = DriverManager.getConnection("....");
%>
<jrun:sql connection="<%= con %>" id="q1">
select * from Table1
</jrun:sql>

<jrun:sql driver="...." url="...." id="q2">
select * from Table1
</jrun:sql>

<%-- sql uses java:comp/env/jdbc/dsn1 to lookup a datasource --%>

<jrun:sql datasrc="dsn1" id="q3">
select * from Table1
</jrun:sql>

<%-- you can enumerate the QueryTable by: --%>

<jrun:param id="q3" type="QueryTable"/>
<jrun:foreach item="x" group="<%= q3.Names %>">
<%= x %><br>
</jrun:foreach>

<jrun:foreach group="<%= q3 %>">
<jrun:foreach item="y" group="<%= q3.Values %>">
<%= y %><br>
</jrun:foreach>
</jrun:foreach>

<%-- OR --%>

<jrun:param id="q3" type="QueryTable"/>
<jrun:foreach group="<%= q3 %>">
<%
int count = q3.getColumnCount();
for (int i = 0;i < count;i += 1) {
%>
<%= q3.get(i) %><br>
<%
}
%>
</jrun:foreach>

<%-- OR --%>

<jrun:param id="q3" type="QueryTable"/>
<%
```

```
while (q3.next()) {  
    %>  
    <jrun:foreach item="y" group="<% q3.Values %>">  
        <%= y %><br>  
    </jrun:foreach>  
    <%  
    }  
    %>  
  
<%-- OR, if you want to use column names... --%>  
  
<jrun:param id="q3" type="QueryTable"/>  
<jrun:foreach group="<% q3 %>">  
    <%= q3.get("id") %><br>  
    <%= q3.get("lastname") %><br>  
    <%= q3.get("firstname") %><br>  
    </jrun:foreach>  
  
<%-- OR, if you want to use column index... --%>  
  
<jrun:param id="q3" type="QueryTable"/>  
<jrun:foreach group="<% q3 %>">  
    <%= q3.get(1) %><br>  
    <%= q3.get(2) %><br>  
    <%= q3.get(3) %><br>  
    </jrun:foreach>
```

SqlParam

Syntax <sqlparam ... />

Usage The `sqlparam` tag works with `sql` to dynamically construct parameterized SQL statements. This tag is very useful for objects (without textual representation) that must be inserted into the SQL statement enclosed by `sql` tags.

Attributes **value**

Required. Takes `java.lang.Object` as an attribute. The object represents a SQL parameter value.

sqltype

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int` as an attribute. The integer values that `sqlparam` supports come from the supported SQL types in `java.sql.Types`. If a string is specified, it must be one of the constant strings in `java.sql.Types`. Valid string values are:

ARRAY, BIGINT, BINARY, BIT, BLOB, CHAR, CLOB, DATE, DECIMAL, DISTINCT, DOUBLE, FLOAT, INTEGER, JAVA_OBJECT, LONGVARBINARY, LONGVARCHAR, NULL, NUMERIC, OTHER, REAL, REF, SMALLINT, STRUCT, TIME, TIMESTAMP, TINYINT, VARBINARY, VARCHAR.

scale

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int` as an attribute. This represents the number of digits after decimal point. This is only valid when `sqltype = NUMERIC` or `DECIMAL`.

**TagLib
Descriptor**

```
<tag>
  <name>sqlparam</name>
  <tagclass>allaire.taglib.SqlParamTag</tagclass>
  <bodycontent>empty</bodycontent>
  <attribute>
    <name>value</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>sqltype</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>scale</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:sql driver="..." url="..." id="result">
select * from Table1 where id = <jrun:sqlparam value="<%=> ... %>" />
</jrun:sql>
```

SendMsg

Syntax <sendmsg ...>
message body with optional <msgparam ... />...
</sendmsg>

Usage The `sendmsg` tag sends the enclosed text message using the Java Message Service (JMS). It is possible to have flow-control tags within the `sendmsg` tag so that text message can be dynamically constructed. The `sendmsg` tag can also be used with `msgparam` tags to specify additional message properties. If the `sendmsg` tag is within a `transaction` tag, the `sendmsg` will be a transaction participant and the success or failure of the `sendmsg` operation will be based on all the operations within the same transaction. If all the operations (including `sql`, `sendmsg` and `getmsg`) within the `transaction` are successful, all the changes will be committed. Otherwise, an exception will be raised and changes will be rolled back. The `sendmsg` tag uses the `allaire.taglib.Transaction` interface to detect if it is enclosed in a `transaction` tag.

Note The `sendmsg` tag is for point-to-point messaging only.

The `sendmsg` tag supports syntax of using JMS queue connection factory and JMS message queue from JNDI lookup - J2EE standard. JMS Connection pooling is provided. It is the responsibility of the J2EE server administrator to predefine message queue objects for use by Web applications. For details, refer to the J2EE specification and the following example(s).

Attributes **msgsrc**

Required. Takes `java.lang.String` or `javax.jms.QueueConnectionFactory` as an attribute. If a string is specified, it is assumed that the queue connection factory can be obtained by performing a JNDI lookup with "`java:comp/env/jms/[msgsrc]`". If the `QueueConnectionFactory` object cannot be obtained from the default `InitialContext`, it is necessary to use the `jndi` tag to lookup the factory from a different `InitialContext`. Refer to the `jndi` section for details.

queue

Required. Takes `java.lang.String` or `javax.jms.Queue` as an attribute. This attribute represents a message queue object. If a string is specified, it is assumed that the queue can be obtained by performing a JNDI lookup with "`java:comp/env/jms/[queue]`". If the `Queue` object cannot be obtained from the default `InitialContext`, it is necessary to use the `jndi` tag to lookup the queue from a different `InitialContext`. Refer to the `jndi` section for details.

username

Optional. Takes `java.lang.String` as an attribute. This attribute represents a username that is needed for message queue authentication.

password

Optional. Takes `java.lang.String` as an attribute. This attribute represents a password that is needed for message queue authentication.

delivery

Optional. Takes `java.lang.String` as an attribute. This attribute represents the message delivery mode. Valid values are `PERSISTENT` or `NON_PERSISTENT`. By default, `delivery = "PERSISTENT"`.

priority

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int` as an attribute. This attribute represents message priority. The default value is 4.

expire

Optional. Takes `java.lang.String` or `java.lang.Long` or `long` as an attribute. It represents the time interval for the message to expire (in milliseconds). By default, the message never expires.

**TagLib
Descriptor**

```
<tag>
  <name>sendmsg</name>
  <tagclass>allaire.taglib.SendMsgTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>msgsrc</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>queue</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>username</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>password</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>delivery</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>priority</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
```

```
<name>expire</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Interface package allaire.taglib;

public interface SendMsg {
 void setProperty(String name, Object value) throws JspTagException;
}

Example This example shows the most common way of using the `sendmsg` tag. A Web application usually needs to perform a JNDI lookup to obtain a JMS queue connection factory. It also needs to obtain a predefined JMS message queue object using the same approach. The retrieval of these distributed objects can be done by using `jndi` and `param` tags.

First, `param` tag is used to declare the scripting variables that hold the JMS queue connection factory and the JMS message queue. Then `jndi` tag can be used to locate the objects. Finally, `sendmsg` tag can use the returned objects from JNDI to perform the message sending operation.

```
<%@ taglib uri="jruntags" prefix="jrun" %>

<%-- sendmsg uses java:comp/env/jms/QueueConnectionFactory to lookup
a queue connection factory and uses java:comp/env/jms/Queue1 to lookup
a queue. --%>

<jrun:sendmsg msgsrc="QueueConnectionFactory" queue="Queue1">
This is a text message.
</jrun:sendmsg>

<%-- OR, if the factory and the queue can't be found from the
default InitialConext, it is necessary to use jndi to specify
a custom provider class name and url. --%>

<jrun:jndi provider="..." url="..."
    name="java:comp/env/jms/QueueConnectionFactory" id="f"/>
<jrun:jndi provider="..." url="..."
    name="java:comp/env/jms/Queue1" id="q"/>

<jrun:sendmsg msgsrc=<%= page.getAttribute("f") %>
    queue=<%= page.getAttribute("q") %>>
This is a text message.
</jrun:sendmsg>
```

MsgParam

Syntax <msgparam ... />

Usage The msgparam tag is used with the sendmsg tag to specify JMS message properties.

Attributes **name**

Required. Takes java.lang.String as an attribute. This attribute represents the JMS message property name.

value

Required. Takes java.lang.String as an attribute. This attribute represents the JMS message property value.

TagLib Descriptor

```
<tag>
  <name>msgparam</name>
  <tagclass>allaire.taglib.MsgParamTag</tagclass>
  <bodycontent>empty</bodycontent>
  <attribute>
    <name>name</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>value</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>
```

Example For descriptions of this example, refer to the sendmsg section.

```
<%@ taglib uri="jruntags" prefix="jrun" %>

<%-- sendmsg uses java:comp/env/jms/QueueConnectionFactory to lookup
a queue connection factory and uses java:comp/env/jms/Queue1 to lookup
a queue. --%>

<jrun:sendmsg msgsra="QueueConnectionFactory" queue="Queue1">
<jrun:msgparam name="foo" value="bar"/>
This is a text message.
</jrun:sendmsg>
```

GetMsg

Syntax

```
<getmsg ...>
SQL-SELECT filter string...
</getmsg>
```

Usage The `getmsg` tag performs message retrieval from the specified JMS message queue. It takes the enclosed SQL-SELECT string (JMS message selector) to identify the name of the message queue, the search criteria, the order in which the messages are retrieved and the message properties that the `getmsg` tag has to return. It is possible to have flow-control tags within the `getmsg` tag so that the message selector statement can be dynamically constructed. If the `getmsg` tag is within a `transaction` tag, the `getmsg` will be a transaction participant and the success or failure of the `getmsg` operation will be based on all the operations within the same transaction. If all the operations (including `sql`, `sendmsg` and `getmsg`) within the `transaction` are successful, all the changes will be committed. Otherwise, an exception will be raised and changes will be rolled back. The `getmsg` tag uses the `allaire.taglib.Transaction` interface to detect if it is enclosed by a `transaction` tag.

The `getmsg` tag supports syntax of using JMS queue connection factory and JMS message queue from JNDI lookup - J2EE standard. JMS Connection pooling is provided. It is the responsibility of the J2EE server administrator to predefine message queue objects for use by Web applications. For details, refer to the J2EE specification and the example.

The following section illustrates the syntax of the JMS message selector:

```
SELECT {a comma-separated list of property names or *}
FROM {message queue name}
WHERE {a valid JMS message selector string as defined in JMS spec.}
```

The JMS message selector string is OPTIONAL. For details on JMS message selector, refer to `javax.jms.Message` JavaDoc.

Attributes **msgsrc**

Required. Takes `java.lang.String` or `javax.jms.QueueConnectionFactory` as an attribute. If a string is specified, it is assumed that the queue connection factory can be obtained by performing a JNDI lookup with "`java:comp/env/jms/[msgsrc]`". If the `QueueConnectionFactory` object cannot be obtained from the default `InitialContext`, it is necessary to use the `jndi` tag to lookup the factory from a different `InitialContext`. Refer to the `jndi` section for details.

username

Optional. Takes `java.lang.String` as an attribute. This attribute represents a username that is needed for message queue authentication.

password

Optional. Takes `java.lang.String` as an attribute. This attribute represents a password that is needed for message queue authentication.

id

Required. Takes `java.lang.String` as an attribute. The string name is used as the scripting variable name for the message retrieval.

scope

Optional. Takes `java.lang.String` or `int` as an attribute. Valid string values are `page`, `request`, `session` and `application`. The default is "page".

TagLib Descriptor

```
<tag>
  <name>getmsg</name>
  <tagclass>allaire.taglib.GetMsgTag</tagclass>
  <teiclass>allaire.taglib.GetMsgTei</teiclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>msgsrc</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>username</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>password</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>id</name>
    <required>true</required>
    <rteprvalue>false</rteprvalue>
  </attribute>
  <attribute>
    <name>scope</name>
    <required>false</required>
    <rteprvalue>false</rteprvalue>
  </attribute>
</tag>
```

Scripting Variables The `getmsg` tag stores the message retrieval result in the `pageContext` object. If `id` is specified, an instance of `allaire.taglib.MessageTable` will be stored. The scope of this object depends on the `scope` attribute, which is `PageContext.PAGE_SCOPE` by default. The following is the api of `allaire.taglib.MessageTable`. Refer to the `sql` section for information on the `allaire.taglib.Table` API:

```
package allaire.taglib;

public class MessageTable extends Table {
    public MessageTable()
    public String[] Props;
    public Object[] Values;
    public Object Message;
    public void populate(String[] props, MessageConsumer receiver)
        throws JMSException;
    public boolean next();
}
```

Example This example shows the use of the `getmsg` tag.

```
<%@ page import="allaire.taglib.*" %>
<%@ taglib uri="jruntags" prefix="jrun" %>

<jrun:getmsg msgsrc="QueueConnectionFactory" id="table">
select * from Queue1
</jrun:getmsg>

<%-- You can enumerate the JMS messages by using the same technique as
described in sql section. --%>

<jrun:param id="table" type="MessageTable"/>
<jrun:foreach group="<%= table %>">
...
</jrun:foreach>
```

Transaction

Syntax

```
<transaction>
  <sql ...>
    sql statement
  </sql>
  ...more <sql>, <sendmsg> or <getmsg> tags...
</transaction>
```

Usage The `transaction` tag allows JSP developers to write distributed transaction logic in JSP. Distributed transactions are used when data integrity must be maintained, such as when updating more than one database table on two or more machines. In case of failure in one of the database servers, the transaction must rollback the changes made to the others. A message sending operation to a specified message queue could also be transacted, that is, the message will be cancelled if the transaction fails. You usually use the `transaction` tag with the `sql`, `sendmsg` and `getmsg` tags. These tags automatically detect whether they are within a `transaction` tag and implement logic accordingly.

TagLib Descriptor

```
<tag>
  <name>transaction</name>
  <tagclass>allaire.taglib.TransactionTag</tagclass>
  <bodycontent>JSP</bodycontent>
</tag>
```

Interface

```
package allaire.taglib;
public interface Transaction {
    void setSql(String ds, String sql) throws JspTagException;
}
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:transaction>
  <jrun:sql datasrc="s1"> <%-- java:comp/env/jdbc/s1 --%>
    SQL statement 1...
  </jrun:sql>
  <jrun:sql datasrc="s2"> <%-- java:comp/env/jdbc/s2 --%>
    SQL statement 2...
  </jrun:sql>
  <%--
    java:comp/env/jms/QueueConnectionFactory
    java:comp/env/jms/Queue1
  --%>
  <jrun:sendmsg msgssrc="QueueConnectionFactory" queue="Queue1">
    Successful transaction...
  </jrun:sendmsg>
</jrun:transaction>
```

Jndi

Syntax <jndi ... />

Usage The jndi tag allows JSP to access distributed objects in a J2EE server environment. It supports 4 main actions: lookup, list, search and attributes.

Syntax Syntax Type 1: Supports lookup, list and attribute.

Types Syntax Type 2: Supports search.

The following table shows which attributes are available using the two different syntax types:

Jndi Tag Syntax Types		
Attribute	Type 1	Type 2
action	x	x
name	x	x
provider	x	x
url	x	x
id	x	x
scope	x	x
attributes		x

Attributes **action (types 1,2)**

Required. Takes java.lang.String as an attribute. This attribute represents the JNDI action. Valid values are `lookup`, `list`, `attribute`, or `search`. If `action="lookup"`, the tag will return an object from the JNDI server. If `action="list"`, the tag will return a list of objects that matches the prefix name. If `action="attribute"`, the tag will return a list of attributes from the specified directory service (for example, LDAP). If `action="search"`, the tag will return an enumeration of objects bounded to a name from the specified directory service.

name (types 1,2)

Required. Takes `java.lang.String` or `javax.naming.Name` as an attribute. This attribute represents the JNDI lookup/search name. An instance of `javax.naming.Name` can also be used.

provider (types 1,2)

Required. Takes java.lang.String as an attribute. This attribute represents the JNDI lookup/Directory service provider class.

url (types 1,2)

Required. Takes java.lang.String as an attribute. This attribute represents the JNDI lookup/Directory service provider URL.

id (types 1,2)

Required. Takes java.lang.String as an attribute. The string name is used as the scripting variable name for the JNDI lookup.

scope (types 1,2)

Optional. Takes java.lang.String or int as an attribute. Valid string values are page, request, session and application. The default is "page".

attributes (type 2)

Required. Takes java.util.Dictionary, java.util.Map or javax.naming.Attributes as an attribute. This attribute represents a set of attributes to search for. The action "search" in a single context for objects that contain this specified set of attributes and returns all the attributes of such objects.

TagLib Descriptor

```
<tag>
<name>jndi</name>
<tagclass>allaire.taglib.JndiTag</tagclass>
<teiclass>allaire.taglib.JndiTei</teiclass>
<bodycontent>empty</bodycontent>
<attribute>
    <name>id</name>
    <required>true</required>
    <rteprvalue>false</rteprvalue>
</attribute>
<attribute>
    <name>scope</name>
    <required>false</required>
    <rteprvalue>false</rteprvalue>
</attribute>
<attribute>
    <name>action</name>
    <required>true</required>
    <rteprvalue>false</rteprvalue>
</attribute>
<attribute>
    <name>name</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
</attribute>
<attribute>
    <name>provider</name>
    <required>false</required>
```

```
<rteprvalue>true</rteprvalue>
</attribute>
<attribute>
  <name>attributes</name>
  <required>false</required>
  <rteprvalue>true</rteprvalue>
</attribute>
<attribute>
  <name>url</name>
  <required>false</required>
  <rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Scripting Variables The type of objects created by the jndi tag depends on the JNDI action. The scope of this variable depends on the scope attribute, which is PAGE_SCOPE by default.

The following is the API of NameTable, DirSearchTable and AttributeTable. Refer to sql for the API of allaire.taglib.Table:

```
package allaire.taglib;

public class NameTable extends Table {
    public NameTable()
    public String Name;
    public String ClassName;
    public boolean isRelative;
    public void populate(NamingEnumeration enum) throws NamingException;
    public boolean next();
}

public class DirSearchTable extends Table {
    public DirSearchTable()
    public String Name;
    public String ClassName;
    public boolean isRelative;
    public AttributeTable Attributes;
    public void populate(NamingEnumeration enum)
        throws NamingException;
    public boolean next();
}

public class AttributeTable extends Table {
    public AttributeTable()
    public String ID;
    public Attribute Attribute;
    public void populate(Attributes attrs) throws NamingException;
    public boolean next();
}
```

Example

```
<%@ page import="allaire.taglib.*" %>
<%@ taglib uri="jruntags" prefix="jrun" %>

<jrun:jndi action="lookup" name="java:comp/UserTransaction" id="txn"/>
<jrun:jndi action="list" name="java:comp/env/jdbc" id="enum"/>
```

SendMail

Syntax

```
<sendmail ...>
  mail body with optional <mailparam ... />
</sendmail>
```

Usage The `sendmail` tag constructs multipart e-mails (with attachments) and sends them to the specified mail server. It is possible to have flow-control tags within the `sendmail` tag so that the e-mail body can be dynamically constructed. The `sendmail` tag can also be used with `mailparam` tags to include e-mail attachments.

The `sendmail` tag supports two types of syntax. The first type allows JSP developers to specify their own mail server information. The second type requires the J2EE server administrator to predefine JavaMail session objects for use by Web applications. For details, refer to the J2EE specification and the following example.

Syntax Types **Syntax Type 1:** Mail server properties are provided.
Syntax Type 2: JavaMail session object is provided.

The following table shows which attributes are available using the two different syntax types:

SendMail Tag Syntax Types		
Attribute	Type 1	Type 2
host	x	
port	x	
timeout	x	
session		x
sender	x	x
recipient	x	x
cc	x	x
bcc	x	x
subject	x	x

Attributes host (type 1)

Required. Takes `java.lang.String` as an attribute. This attribute represents mail server hostname.

port (type 1)

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int`. This attribute represents the mail server port number. If a string is specified, it will be parsed to an integer value. The default is 25.

timeout (type 1)

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int`. This attribute represents mail server timeout value in milliseconds. If a string is specified, it will be parsed to an integer value. The default is 3000.

session (type 2)

Required. Takes `java.lang.String` or `javax.mail.Session`. This attribute represents a JavaMail session object. If a string is specified, it is assumed that the mail session object can be obtained by performing a JNDI lookup with "`java:comp/env/mail/[session]`". If the Session object cannot be obtained from the default `InitialContext`, it is necessary to use the `jndi` tag to lookup the session from a different `InitialContext`. Refer to the `jndi` section for details. Usually, JavaMail session objects should be pre installed by J2EE server administrators.

sender (types 1,2)

Required. Takes `java.lang.String` or `javax.mail.internet.InternetAddress` as an attribute. This represents the sender's e-mail address.

recipient (types 1,2)

Required. Takes `java.lang.String`, `java.lang.String[]`, `java.util Enumeration`, `java.util.Iterator`, `javax.mail.internet.InternetAddress` or `javax.mail.internet.InternetAddress[]` as an attribute. This attribute represents the recipient's e-mail addresses. If a string is specified, it will check whether the string is a comma-separated e-mail address list.

cc (types 1,2)

Optional. Takes `java.lang.String`, `java.lang.String[]`, `java.util Enumeration`, `java.util.Iterator`, `javax.mail.internet.InternetAddress` or `javax.mail.internet.InternetAddress[]` as an attribute. This attribute represents the carbon copy e-mail addresses. If a string is specified, it will check whether the string is a comma-separated e-mail address list.

bcc (types 1,2)

Optional. Takes `java.lang.String`, `java.lang.String[]`, `java.util.Enumeration`, `java.util.Iterator`, `javax.mail.internet.InternetAddress` or `javax.mail.internet.InternetAddress[]` as an attribute. This attribute represents the blind carbon copy e-mail addresses. If a string is specified, it will check whether the string is a comma-separated e-mail address list.

subject (types 1,2)

Optional. Takes `java.lang.String` as an attribute. This attribute represents the e-mail subject line.

TagLib Descriptor

```
<tag>
  <name>sendmail</name>
  <tagclass>allaire.taglib.SendMailTag</tagclass>
  <teiclass>allaire.taglib.SendMailTei</teiclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>host</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>port</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>timeout</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>session</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>sender</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>recipient</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>cc</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
```

```
<attribute>
  <name>bcc</name>
  <required>false</required>
  <rteprvalue>true</rteprvalue>
</attribute>
<attribute>
  <name>subject</name>
  <required>false</required>
  <rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Interface package allaire.taglib;

```
public interface SendMail {
    void setBody(String text);
    void setAttachment(URL url);
    void setHeader(String name, String value);
}
```

Example This example shows two different ways of using `sendmail` to send e-mail messages. For examples on sending e-mail attachments, refer to the `mailparam` section.

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:sendmail host="..." sender="..." recipient="...">
Hello world.
</jrun:sendmail>

<%-- java:comp/env/mail/session1 --%>

<jrun:sendmail session="session1" sender="..." recipient="...">
Hello world.
</jrun:sendmail>
```

MailParam

Syntax <mailparam ... />

Usage The `mailparam` tag is used with `sendmail` tag to dynamically attach messages/files into the main e-mail body from some specified URLs in `mailparam`. It can also be used to specify additional e-mail headers.

Syntax Syntax Type 1: Supports e-mail attachments.

Types Syntax Type 2: Supports additional e-mail headers.

The following table shows which attributes are available using the two different syntax types:

MailParam Tag Syntax Types		
Attribute	Type 1	Type 2
attachurl	x	
name		x
value		x

Attributes **attachurl (type 1)**

Required. Takes `java.lang.String` as an attribute. This attribute represents the URL to the document that will be included in the MIME message (e-mail attachment). The URL can be absolute (starts with `/`) or relative to the page using this tag.

name (type 2)

Required. Takes `java.lang.String` as an attribute. This attribute represents the additional e-mail header name.

value (type 2)

Required. Takes `java.lang.String` as an attribute. This attribute represents the additional e-mail header value.

TagLib Descriptor

```
<tag>
  <name>mailparam</name>
  <tagclass>allaire.taglib.MailParamTag</tagclass>
  <teiclass>allaire.taglib.MailParamTei</teiclass>
  <bodycontent>empty</bodycontent>
  <attribute>
    <name>attachurl</name>
    <required>false</required>
    <rtextprvalue>true</rtextprvalue>
  </attribute>
  <attribute>
    <name>name</name>
    <required>false</required>
    <rtextprvalue>true</rtextprvalue>
  </attribute>
  <attribute>
    <name>value</name>
    <required>false</required>
    <rtextprvalue>true</rtextprvalue>
  </attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:sendmail host="..." sender="..." recipient="...">
Hello world.
<jrun:mailparam attachurl="allaire.gif" />
</jrun:sendmail>
```

GetMail

Syntax <getmail ...>
SQL-SELECT e-mail retrieval string.
</getmail>

Usage The getmail tag performs e-mail message retrieval from the specified mail server. Takes the enclosed SQL SELECT string to identify the name of the e-mail folder, the search criteria, the order in which the messages are retrieved and the e-mail headers/content that the getmail tag has to return. It is possible to have flow-control tags within the getmail tag so that the SELECT statement can be dynamically constructed.

The following illustrates the syntax of the getmail e-mail retrieval string:

```
SELECT {a comma-separated list of mail headers or *}  
FROM {mail folder name}  
WHERE {optional search terms available in JavaMail API}
```

The following list shows the search terms available:

Valid Search Terms for getmail Retrieval String		
Search Term	Valid Operators	Data Type/format
Sender	=, contains	String literal
Recipient	=, contains	String literal
Cc	=, contains	String literal
Bcc	=, contains	String literal
MessageID	=, contains	String literal
MessageNumber	=, >=, >, <=, <, !=	integer
Subject	=, contains	String literal
SentDate	=, >=, >, <=, <, !=	MM/dd/yyyy hh:mm:ss
ReceivedDate	=, >=, >, <=, <, !=	MM/dd/yyyy hh:mm:ss
Size	=, >=, >, <=, <, !=	Integer

Valid Search Terms for getmail Retrieval String (continued)		
Search Term	Valid Operators	Data Type/format
Flag	=, !=	One the following: <ul style="list-style-type: none"> • ANSWERED • DELETED • DRAFT • FLAGGED • RECENT • SEEN • USER
Body	contains	String literal

The logical operators "and", "or", "not" can be used to combine search terms with the use of parenthesis "(" and ")", for example:

```
SELECT * FROM InBox
WHERE (Sender contains 'allaire.com' or
       Sender contains 'sun.com')
      and Size < 1000
```

The `getmail` tag supports two types of syntax. The first type allows JSP developers to specify their own mail server information. The second type requires the J2EE server administrator to predefine JavaMail session objects for use by Web applications. For details, refer to the J2EE specification and the following example(s).

- Syntax Types**
- Syntax Type 1:** Mail server properties are provided.
 - Syntax Type 2:** JavaMail session object is provided.

The following table shows which attributes are available using the two different syntax types:

GetMail Tag Syntax Types		
Attribute	Type 1	Type 2
host	x	
port	x	
timeout	x	
session		x
username	x	x

GetMail Tag Syntax Types (continued)		
Attribute	Type 1	Type 2
password	x	x
id	x	x
scope	x	x
protocol	x	x

Attributes host (type 1)

Required. Takes `java.lang.String` as an attribute. This attribute represents mail server hostname.

port (type 1)

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int`. This attribute represents mail server port number. If a string is specified, it will be parsed to an integer value.

timeout (type 1)

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int`. This attribute represents mail server timeout value in milliseconds. If a string is specified, it will be parsed to an integer value. The default is 3000.

session (type 2)

Required. Takes `java.lang.String` or `javax.mail.Session`. This attribute represents a JavaMail session object. If a string is specified, it is assumed that the mail session object can be obtained by doing a JNDI lookup with "java:comp/env/mail/[session]". If the Session object cannot be obtained from the default InitialContext, it is necessary to use the `jndi` tag to lookup the session from a different InitialContext. Refer to the `jndi` section for details. Usually, JavaMail session objects should be pre installed by J2EE server administrators.

username (type 1,2)

Required. Takes `java.lang.String` as an attribute. This attribute represents a username that is needed for e-mail message retrieval.

password (type 1,2)

Required. Takes `java.lang.String` as an attribute. This attribute represents a password that is needed for e-mail message retrieval.

id (type 1,2)

Required. Takes `java.lang.String` as an attribute. The string name is used as the scripting variable name for the message retrieval.

scope (type 1,2)

Optional. Takes `java.lang.String` or `int` as an attribute. Valid string values are `page`, `request`, `session` and `application`. The default is `page`.

protocol (type 1,2)

Required. Takes `java.lang.String` as an attribute. This attribute represents the mail protocol the `getmail` tag uses. Valid values are `imap` and `pop3`.

TagLib Descriptor

```
<tag>
  <name>getmail</name>
  <tagclass>allaire.taglib.GetMailTag</tagclass>
  <teiclass>allaire.taglib.GetMailTei</teiclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>host</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>port</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>timeout</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>session</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>id</name>
    <required>true</required>
    <rteprvalue>false</rteprvalue>
  </attribute>
  <attribute>
    <name>scope</name>
    <required>false</required>
    <rteprvalue>false</rteprvalue>
  </attribute>
  <attribute>
    <name>username</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>password</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>
```

```
<attribute>
    <name>protocol</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Scripting Variables

The `getmail` tag stores the e-mail message retrieval result in the `pageContext` object. If `id` is specified, an instance of `allaire.taglib.e-mailTable` will be stored. The scope of this object depends on the `scope` attribute, which is `PageContext.PAGE_SCOPE` by default. The following is the api of `allaire.taglib.e-mailTable`. Refer to the `sql` section for the `allaire.taglib.Table` api:

```
package allaire.taglib;

public class e-mailTable extends Table {
    public e-mailTable()
    public String[] Headers;
    public String[] Values;
    public Object Message;
    public void populate(String[] fields, javax.mail.Message[] msgs)
        throws MessagingException;
    public boolean next();
}
```

Example

This example shows two different ways of using `getmail` to retrieve e-mail messages.

```
<%@ page import="allaire.taglib.*" %>
<%@ taglib uri="jruntags" prefix="jrun" %>

<jrun:getmail host="..." username="..." password="..." id="e-mails"
    protocol="imap">
    select * from Inbox/People/Colton where Subject contains 'jrun'
</jrun:getmail>

<%-- java:comp/env/mail/session1 --%>

<jrun:getmail session="session1" id="e-mails" protocol="pop3">
    select * from InBox where Subject contains 'jrun'
</jrun:getmail>

<jrun:param id="e-mails" type="e-mailTable"/>
<jrun:foreach group="page.e-mails">
    ...<%= e-mails.get("...") %><br>
</jrun:foreach>
```

Servlet

Syntax < servlet ...>
optional < servletparam ... />
</ servlet>

Usage The `servlet` tag collects servlet attributes from `servletparam` and uses `RequestDispatcher` interface to invoke the specified servlet. The output of the invoked servlet is then embedded into the calling JSP.

Attributes **code**
Required. Takes `java.lang.String` as an attribute. This attribute represents the target servlet name.

TagLib Descriptor <tag>
<name>servlet</name>
<tagclass>allaire.taglib.ServletTag</tagclass>
<bodycontent>JSP</bodycontent>
<attribute>
<name>code</name>
<required>true</required>
<rteprvalue>true</rteprvalue>
</attribute>
</tag>

Interface package allaire.taglib;

public interface Servlet {
 void setServletParam(String name, Object value);
 String getCode();
}

Example <%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:servlet code="SnoopServlet">
</jrun:servlet>

ServletParam

Syntax <servletparm ... />

Usage The `servletparm` tag works with `servlet` tag to specify servlet request attributes.

Attributes **name**

Required. Takes `java.lang.String` as an attribute. This attribute represents the target servlet attribute name.

value

Required. Takes `java.lang.Object` as an attribute. This attribute represents the target servlet attribute value.

TagLib <tag>

Descriptor

```
<name>servletparm</name>
<tagclass>allaire.taglib.ServletParamTag</tagclass>
<bodycontent>empty</bodycontent>
<attribute>
    <name>name</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
</attribute>
<attribute>
    <name>value</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:servlet code="SnoopServlet">
<jrun:servletparm name="key" value="<% new Hashtable() %>" />
</jrun:servlet>
```

Query2Xml

Syntax <query2xml ... />

Usage The query2xml tag performs simple conversion of tabular data into XML format.

Attributes **query**

Required. Takes java.lang.String, java.sql.ResultSet, javax.sql.RowSet or allaire.taglib.Table. If a string is specified, it is assumed that the query result object can be obtained by invoking pageContext.getAttribute().

id

Optional. Takes java.lang.String. A variable name to reference a XML document object or a reader object. If id is not specified, the generated XML output will be embedded into the calling JSP.

type

Optional. Takes java.lang.String. The type of XML object this tag generates. Valid values are DOM and TEXT. If DOM is specified, an instance of org.w3c.dom.Document will be created. If TEXT is specified, an instance of java.io.BufferedReader will be created. The default is TEXT.

scope

Optional. Takes java.lang.String.id scope. Available values are page, request, session and application. The default is page.

rootname

Optional. Takes java.lang.String. Specifies the query tag name. The default is "table".

rowname

Optional. Takes java.lang.String. Specifies the row tag name. The default is "row".

**TagLib
Descriptor**

```
<tag>
  <name>query2xml</name>
  <tagclass>allaire.taglib.Query2XmlTag</tagclass>
  <teiclass>allaire.taglib.Query2XmlTei</teiclass>
  <bodycontent>empty</bodycontent>
  <attribute>
    <name>query</name>
    <required>true</required>
    <rtextrvalue>true</rtextrvalue>
  </attribute>
  <attribute>
    <name>id</name>
    <required>false</required>
    <rtextrvalue>false</rtextrvalue>
```

```
</attribute>
<attribute>
    <name>scope</name>
    <required>false</required>
    <rtextrvalue>false</rtextrvalue>
</attribute>
<attribute>
    <name>type</name>
    <required>false</required>
    <rtextrvalue>false</rtextrvalue>
</attribute>
<attribute>
    <name>rootname</name>
    <required>false</required>
    <rtextrvalue>true</rtextrvalue>
</attribute>
<attribute>
    <name>rowname</name>
    <required>false</required>
    <rtextrvalue>true</rtextrvalue>
</attribute>
</tag>
```

Scripting Variable If id is specified, an instance of org.w3c.dom.Document or java.io.BufferedReader will be stored in the pageContext object, based on the type attribute. The scope of this object depends on the scope attribute, which is PAGE_SCOPE by default.

Example

```
<%@ page contentType="text/xml" %><?xml version="1.0"?>
<%@ taglib uri="jruntags" prefix="jrun" %>

<jrun:sql driver="..." url="..." id="rs" scope="session">
select * from Table1
</jrun:sql>

<jrun:query2xml query="session.rs" />
```

Xslt

Syntax 1 <xslt ...>
xml input...
</xslt>

Usage The xslt tag performs XSL transformations using the enclosed XML input.

Attributes **xsl**

Required. Takes java.lang.String. URL to the XSL. If a string is specified, the URL string can be absolute (starts with "/") or relative to the page using this tag. If a full URL is used (<http://....>), the java.net.URL object must be used.

id

Optional. Takes java.lang.String. The transformation output name.

scope

Optional. Takes java.lang.String. id. Specifies the scope. Available values are page, request, session, or application. The default is page.

Syntax 2 <xslt ... />

The xslt tag performs XSL transformations by taking the XML input from a URL.

xml

Optional. Takes java.lang.String. The URL to the XML. If a string is specified, the URL string can be absolute (starts with /) or relative to the page using this tag. If a full URL is used (<http://....>), the java.net.URL object must be used.

xsl

Required. Takes java.lang.String. The URL to the XSL. If a string is specified, the URL string can be absolute (starts with /) or relative to the page using this tag. If a full URL is used (<http://....>), the java.net.URL object must be used.

id

Optional. Takes java.lang.String. The transformation output name.

scope

Optional. Takes java.lang.String. id. Specifies the scope. Available values are page, request, session, application. The default is page.

TagLib Descriptor

```
<tag>
  <name>xslt</name>
  <tagclass>allaire.taglib.XsltTag</tagclass>
  <teiclass>allaire.taglib.XsltTei</teiclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>xsl</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>id</name>
    <required>false</required>
    <rteprvalue>false</rteprvalue>
  </attribute>
  <attribute>
    <name>scope</name>
    <required>false</required>
    <rteprvalue>false</rteprvalue>
  </attribute>
  <attribute>
    <name>xml</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>
```

Scripting Variable If id is specified, an instance of java.io.BufferedReader will be stored in the pageContext object. The scope of this object depends on the scope attribute, which is PAGE_SCOPE by default.

Example The xslt syntax allows JSP developers to markup the XML data in the JSP for transformation, for example, output from query2xml could be used directly in xslt without creating extra scripting variable. For those who do not prefer embedding XML data into JSP, the empty tag syntax (URL to both XML and XSL) meets their requirements.

```
<%@ taglib uri="jruntags" prefix="jrun" %>

<jrun:xslt xml='<%= new URL("http://.../article.xml") %>' 
xsl="format.xsl"/>

<jrun:sql driver="..." url="..." id="rs">
select * from Table1
</jrun:sql>

<jrun:xslt xsl="format2.xsl">
<jrun:query2xml query="page.rs"/>
</jrun:xslt>
```

Form

Syntax <form ...>
...
(optional) <input ... /> | <select ...> ... </select>
...
</form>

Usage The `form` tag extends the HTML `form` tag by providing client-side JavaScript form validation. This tag also supports existing HTML 4.0 `form` tag attributes.

Attributes

name

Required. Takes `java.lang.String` as an attribute. This attribute represents the HTML form name.

action

Optional. Takes `java.lang.String` as an attribute. This attribute represents the HTML form action. A valid URL is required.

onSubmit

Optional. Takes `java.lang.String` as an attribute. This attribute represents a JavaScript function name. When the submit button is clicked, this JavaScript function will be executed.

other HTML 4.0 attributes...

Any other HTML 4.0 attributes you specify are added as attributes of the generated `form` tag.

**TagLib
Descriptor**

```
<tag>
  <name>form</name>
  <tagclass>allaire.taglib.FormTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>name</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>action</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>onSubmit</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>
```

```
Interface package allaire.taglib;

public interface Form {
    void setJavascriptFunction(String js);
    void useOnError();
    void useHasValue();
    void useCheckNumber();
    void useCheckRange();
    void useCheckDate();
    void useCheckDay();
    void useCheckInteger();
    void useCheckTime();
    void useCheckEuroDate();
    void useCheckPhone();
    void useCheckZip();
    void useCheckCreditCard();
    void useCheckSSC();
}
```

Example <%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:form name="form1" action="form.jsp">
<input type="submit" value="submit">
</jrun:form>

Input

Syntax <input ... />

Usage Used inside the `form` tag to place radio buttons, checkboxes, or text boxes. Provides input validation for the specified control type.

The `input` tag supports the JavaScript `onClick` event in the same manner as the HTML `INPUT` tag:

Syntax Type 1: Built-in validation.

Syntax Type 2: User-defined JavaScript validation.

The following table shows which attributes are available using the two different syntax types:

Input Tag Syntax Types		
Attribute	Type 1	Type 2
name	x	x
type	x	x
value	x	x
required	x	x
onError	x	x
onValidate		x

Attributes **name (type 1,2)**

Required. Takes `java.lang.String`. The UI control name.

type (types 1,2)

Optional. Takes `java.lang.String`. The UI control type. Valid types are checkbox, radio, password, creditcard, date, eurodate, float, integer, ssc, phone, time, zipcode and text. The default is text. If `onValidate` is used, type must be text.

value (types 1,2)

Optional. Takes `java.lang.String`. The field value.

required (types 1,2)

Optional. Takes `java.lang.String`, `java.lang.Boolean`, or `boolean`. If this attribute is specified, a checkbox must be checked, a radio button must be selected, a password must be entered, or a string must be entered in the input field box. The default is false.

onError (types 1,2)

Optional. Takes `java.lang.String`. A JavaScript function name. This JavaScript function is executed if client-side form validation fails.

onValidate (type 2)

Optional. Takes `java.lang.String`. A JavaScript function name. This custom JavaScript function is executed for validating this input field.

other HTML 4.0 attributes...

Any other HTML 4.0 attributes you specify are added as attributes of the generated `input` tag.

TagLib Descriptor

```
<tag>
  <name>input</name>
  <tagclass>allaire.taglib.InputTag</tagclass>
  <bodycontent>empty</bodycontent>
  <info>Input Tag</info>
  <attribute>
    <name>name</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>type</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>value</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>required</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>onError</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>onValidate</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>

<%-- Built-in phone format validation --%>

<jrun:form name="form1" action="input.jsp">
<jrun:input name="input1" type="phone" required="true" />
<input type="submit" value="validate">
</jrun:form>

<%-- Custom input field validation and error message --%>

<script language="javascript">
<!--
function customValidate(obj_value) {
    if (!(obj_value.toString() == "PROMOTION")) {
        return false;
    }
    return true;
}
function customErrorMsg(form_obj, input_obj, obj_value, error_msg) {
    alert("custom error message: Invalid Promotion Code: "+obj_value);
    return false;
}
//-->
</script>

<jrun:form method="post" name="form1" action="form4.jsp">
<jrun:input name="t1" required="true" onValidate="customValidate"
    onError="customErrorMsg" />
<jrun:input name="t2" type="password" required="true"/>
<input type="submit" value="Submit"/>
</jrun:form>
```

Select

Syntax <select ...>...</select>

Usage The `select` tag can take a table of key-value pairs (for example, hashtable) and automatically populate a drop-down list box. It also can take a database query result. In this case, two fields from the query must be specified as the value string and the display string.

Syntax Syntax Type 1: Uses a hashtable.

Types Syntax Type 2: Uses two specified columns in a table.

The following table shows which attributes are available using the two different syntax types:

Select Tag Syntax Types		
Attribute	Type 1	Type 2
name	x	x
size	x	x
hashtable	x	
query		x
value		x
display		x
required	x	x
onError	x	x
selected	x	x

Attributes **name (types 1,2)**

Required. Takes `java.lang.String`. The UI control name.

size (types 1,2)

Optional. Takes `java.lang.String`, `java.lang.Integer` or `int`. The number of rows displayed in this drop-down list box. The default is 1. If a string is specified, the string will be parsed into an integer value.

hashtable (type 1)

Required. Takes subclasses of `java.util.Dictionary` or `java.util.Map`. A table of key and value pairs.

query (type 2)

Required. Takes `java.sql.ResultSet`, `javax.sql.RowSet`, `allaire.taglib.Table` or `java.lang.String`. A database query result. This attribute supports string values for simplicity. The tag will retrieve the query object by invoking: `pageContext.getAttribute(query);`

value (type 2)

Required. Takes `java.lang.String`. The key value.

display (type 2)

Optional. Takes `java.lang.String`. The display value. The default is `value`.

required (types 1,2)

Optional. Takes `java.lang.String`, `java.lang.Boolean` or `boolean`. If this attribute is specified, a checkbox must be checked, a radio button must be selected, a password must be entered, or a string must be entered in the input field box. The default is `false`.

onError (types 1,2)

Optional. Takes `java.lang.String`. A JavaScript function name. This JavaScript function is executed if client-side form validation fails.

selected (types 1,2)

Optional. Takes `java.lang.String`. Values to be pre-selected in the drop-down list box (the keys, not the values in the hashtable).

**TagLib
Descriptor**

```
<tag>
  <name>select</name>
  <tagclass>allaire.taglib.SelectTag</tagclass>
  <teiclass>allaire.taglib.SelectTei</teiclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>name</name>
    <required>true</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>size</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
    <name>hashtable</name>
    <required>false</required>
    <rteprvalue>true</rteprvalue>
  </attribute>
  <attribute>
```

```
<name>query</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
<attribute>
<name>required</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
<attribute>
<name>onError</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
<attribute>
<name>value</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
<attribute>
<name>display</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
<attribute>
<name>selected</name>
<required>false</required>
<rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:param id="rs" type="allaire.taglib.QueryTable"/>
<jrun:sql id="rs" driver="..." url="...">
select * from Table1
</jrun:sql>
<jrun:form name="form1" action="select.jsp">
<jrun:select name="box1" required="true" query="<% rs %>" 
    value="ID" display="LastName">
</jrun:select>
<input type="submit" value="validate">
</jrun:form>
```

Param

Syntax <param ... />

Usage The param tag defines a scripting variable in a specified scope (page, request, session, or application), optionally specifying a default value. Variables set by the param tag can be accessed by other custom tags by retrieving from the pageContext object. The param tag does not reset existing scripting variables.

Note This tag must be a top-level tag.

Attributes

id

Required. Takes java.lang.String. Specifies a scripting variable name.

scope

Optional. Takes java.lang.String. Valid values are page, session, request or application.

type

Optional. Takes java.lang.String. Specifies the scripting variable class type.

default

Optional. Takes java.lang.Object. Specifies the default value of this scripting variable.

**TagLib
Descriptor**

```
<tag>
<name>param</name>
<tagclass>allaire.taglib.ParamTag</tagclass>
<teiclass>allaire.taglib.ParamTei</teiclass>
<bodycontent>empty</bodycontent>
<attribute>
    <name>id</name>
    <required>true</required>
    <rtpvalue>false</rtpvalue>
</attribute>
<attribute>
    <name>scope</name>
    <required>false</required>
    <rtpvalue>false</rtpvalue>
</attribute>
<attribute>
    <name>type</name>
    <required>false</required>
    <rtpvalue>false</rtpvalue>
</attribute>
```

```
<attribute>
  <name>default</name>
  <required>false</required>
  <rteprvalue>true</rteprvalue>
</attribute>
</tag>
```

Scripting Variables If id is specified and scope is page, a variable of the type specified by the type attribute will be created.

Scripting variable information follows:

- Name: TagData.getAttribute("id")
- Type: TagData.getAttribute("type")
- Declare: true
- Scope: AT_END

Example The param tag *must* be a top-level tag so the scripting variable it generates is visible by all the other custom actions in the same page.

```
<%@ taglib uri="jruntags" uri="jrun" %>

<%-- Declares an Integer object and stores it in the session scope. --%>
<jrun:param id="x" type="java.lang.Integer" default="<% new Integer(1) %>" scope="session"/>
<%= session.getAttribute("x") %>

<%-- Declares a Date object and stores it in the page scope. --%>
<jrun:param id="y" type="java.util.Date" default="<% new Date() %>" scope="page"/>
<%= y %>
```

ForEach

Syntax

```
<foreach ... >
...
</foreach>
```

Usage The foreach tag loops over a collection of objects defined by the group attribute. The advantage of using the foreach tag is that it automatically recognizes the enumeration type (`java.util.Enumeration`, `javax.naming.NamingEnumeration`, `java.sql.ResultSet`, `javax.sql.RowSet`, `java.util.Iterator`, `allaire.taglib.Table`, `java.lang.Object[]`) so JSP developers need not memorize different enumeration methods for different collection structures. The `item` and `type` attributes are used when the collection uses one of the following types:

- `java.util Enumeration`
- `javax.naming NamingEnumeration`
- `java.util Iterator`
- `java.lang Object[]`

In this case, a scripting variable is created to hold the object returned from the specified collection.

Syntax Types Syntax Type 1: Item object is returned from the specified collection.
Syntax Type 2: Accesses data in the enumeration directly.

The following table shows which attributes are available using the two different syntax types:

ForEach Tag Syntax Types		
Attribute	Type 1	Type 2
item	x	
type	x	x
group	x	x

Attributes item (type 1)

Optional. Takes `java.lang.String`. This attribute is necessary when the group attribute is of type `java.util.Enumeration`, `javax.naming.NamingEnumeration`, `java.util.Iterator` or `java.lang.Object[]`.

type (type 1,2)

Optional. Takes `java.lang.String`. If item is specified, this attribute represents the class name used to create the item loop scripting variable. The default is `java.lang.Object`.

group (type 1,2)

Required. Takes `java.util.Enumeration`, `javax.naming.NamingEnumeration`, `java.sql.ResultSet`, `javax.sql.RowSet`, `java.util.Iterator`, `allaire.taglib.Table`, `java.lang.Object[]`, or `java.lang.String`. If a string is specified, it is assumed that the enumeration object can be found by invoking `pageContext.getAttribute()`.

TagLib Descriptor

```
<tag>
  <name>foreach</name>
  <tagclass>allaire.taglib.ForEachTag</tagclass>
  <teiclass>allaire.taglib.ForEachTei</teiclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>item</name>
    <required>false</required>
    <rtpexprvalue>false</rtpexprvalue>
  </attribute>
  <attribute>
    <name>type</name>
    <required>false</required>
    <rtpexprvalue>false</rtpexprvalue>
  </attribute>
  <attribute>
    <name>group</name>
    <required>true</required>
    <rtpexprvalue>true</rtpexprvalue>
  </attribute>
</tag>
```

Scripting Variables If `item` is specified, the attribute value will be used as the element variable name.
Scripting variable information follows:

- Name: `TagData.getAttribute("item")`
- Type: `TagData.getAttribute("type")`
- Declare: `true`
- Scope: NESTED

Example

```
<%@ page import="allaire.taglib.*" %>
<%@ taglib uri="jruntags" prefix="jrun" %>

<%-- Syntax 1 --%>
<jrun:foreach item="c" type="Cookie"
               group="<% request.getCookies() %>">
<%= c.getname() %>
</jrun:foreach>

<%-- Syntax 2 --%>
<%-- java:comp/env/jdbc/dsn1 --%>

<jrun:sql datasrc="dsn1" id="rs">
SELECT * FROM Table1
</jrun:sql>

<jrun:param id="rs" type="QueryTable"/>
<jrun:foreach group="<% rs %>">
... <%= rs.get("column_name") %> ...
</jrun:foreach>
```

If

Syntax <if ...>
</if>

Usage The if tag evaluates the expression specified in the expr attribute and conditionally executes the enclosing code block based on the expression result.

Attributes **expr**

Optional. Takes java.lang.String, java.lang.Boolean or boolean. The default is true. Valid string values are true, false, TRUE and FALSE.

TagLib Descriptor

```
<tag>
    <name>if</name>
    <tagclass>allaire.taglib.IfTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <attribute>
        <name>expr</name>
        <required>false</required>
        <rteprvalue>true</rteprvalue>
    </attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:if expr="<%=" true".equals(request.getParameter("foo")) %>">
    foo = true<br>
</jrun:if>
```

Switch

Syntax <switch>
 one or more <case> tags...
 </switch>

Usage The `switch` tag performs conditional block execution embedded by the `case` tags. This tag executes the first case block whose expression evaluates to `true`. Each `case` tag within `switch` tag can optionally have its own expression evaluation.

TagLib <tag>
Descriptor <name>switch</name>
 <tagclass>allaire.taglib.SwitchTag</tagclass>
 <bodycontent>JSP</bodycontent>
 </tag>

Interface package allaire.taglib;

```
public interface Switch {  
    boolean eval(boolean b);  
}
```

Example <%@ taglib uri="jruntags" prefix="jrun" %>
 <jrun:switch>
 <jrun:case expr="...">

 </jrun:case>
 <jrun:case expr="...">

 </jrun:case>
 <jrun:case>

 </jrun:case>
 </jrun:switch>

Case

Syntax <case ...>
...
</case>

Usage The case tag is used within a switch tag and conditionally executes the encapsulated code based on the expression evaluation. The expr attribute is optional. The default is true. A case tag must be within switch tag or else an exception will be thrown. The switch tag executes the first case block whose expression evaluates to true. Refer to the switch tag section for details.

Attributes **expr**
Optional. Takes java.lang.String, java.lang.Boolean or boolean. The default is true. Valid string values are true, false, TRUE and FALSE.

TagLib Descriptor

```
<tag>
    <name>case</name>
    <tagclass>allaire.taglib.CaseTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <attribute>
        <name>expr</name>
        <required>false</required>
        <rteprvalue>true</rteprvalue>
    </attribute>
</tag>
```

Example

```
<%@ taglib uri="jruntags" prefix="jrun" %>
<jrun:switch>
<jrun:case expr="...">
...
</jrun:case>
<jrun:case expr="...">
...
</jrun:case>
<jrun:case>
...
</jrun:case>
</jrun:switch>
```

